ISSN 1410 - 9158

Majalah Ilmiah
Informatika, Komputer, dan Bisnis

FORMAT

Volume 10, Nomor 1, Januari 2009

SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER AKAKOM
YOGYAKARTA

Majalah Ilmiah
Informatika, Komputer, dan Bisnis
FORMAT

Majalah Ilmiah FORMAT diterbitkan empat bulan sekali oleh
STMIK AKAKOM dengan ISSN 1410 - 9158

ALAMAT REDAKSI:
STMIK AKAKOM
Jl. Raya Janti, *Ring Road* Timur, Yogyakarta 55198
Telepon: +62-274-486664
Faksimile: +62-274-486438 E-mail: format@netexecutive.com

Dari Redaksi

Kami panjatkan puji dan syukur atas rahmat dan berkah dari Tuhan Yang Maha Esa hingga kami dapat menyelesaikan dan menerbitkan majalah Format pada nomor pertama, tahun kesepuluh. Pada tahun yang kesepuluh ini kami mencoba untuk memperluas cakupan materi dari berbagai hasil penelitian dan karya ilmiah, namun tetap sesuai dengan misinya.

Dalam edisi ini para pembaca akan melihat topik-topik mengenai *Aplikasi Nilai Laporan Praktikum On-Line, Penyelesaian Model Transportasi Bikriteria Dengan Algoritma Genetik, Survei dan Perbandingan Berbagai Software Bebas Pembangun Ontologi, Aplikasi Penyelesaian Integral Rangkap Dua Menggunakan Metode Monte Carlo, Implementasi Advanced Encryption Standard (AES) (RIJNDAEL) Pada Mikrokontroler Atmega 8, Internal Integrated Marketing Strategi Untuk Meraih Keunggulan Bersaing Jangka Panjang,* yang sekiranya akan menarik untuk diulas.

Harapan kami semoga apa yang kami suguhkan kali ini dapat membawa manfaat bagi peminat, dan menambah referensi pembaca pada bidang-bidang tertentu. Terima kasih diucapkan, atas saran dan masukan yang telah kami terima demi kemajuan majalah ilmiah ini. Saran, ide, dan gagasan dari para pembaca tetap kami tunggu untuk perbaikan pada penerbitan edisi mendatang di abad millenium ini.

Daftar Isi

6. Suarga, 2005, *Fisika Komputasi Solusi Problema Fisika dengan Matlab*, Andi, Yogyakarta.

IMPLEMENTASI ADVANCED ENCRYPTION STANDARD (AES) (RIJNDAEL) PADA MIKROKONTROLER ATMEGA 8

Oleh: Totok Budioko

ABSTRAK

Peralatan-peralatan kecil seperti pembaca dan penulis SIMCARD, kunci pintu rahasia, pengendali jarak jauh, penyimpan data, transmisi data melalui gelombang radio membutuhkan kemampuan enkripsi/dekripsi untuk menjamin keamanan data baik pada proses penyimpanan maupun pengiriman. Peralatan-peralatan tersebut dapat direalisasikan menggunakan berbagai mikrokontroler, 8 bit, 16 bit atau 32 bit dengan arsitektur CISC atau RISC.

Implementasi algoritma enkripsi/dekripsi yang optimal pada mikrokontroler 8 bit dipengaruhi oleh jenis mikrokontroler dan algoritma enkripsi/dekripsi yang akan digunakan. Salah satu mikrokontroler yang banyak digunakan dan mudah didapat di pasaran adalah AVR ATMega 8. Sedangkan algoritma enkripsi/dekripsi yang menjadi standar enkripsi adalah Rijndael (AES)

Makalah ini membahas implementasi algoritma enkripsi/dekripsi AES (Rijndael) pada mikrokontroler ATMEL AVR ATMega 8.

Kata Kunci: enkripsi, dekripsi, kunci rahasia, AES, Rijndael, mikrokontroler, ATMega 8

1 PENDAHULUAN

1.1 Latar Belakang

Mikrokontroler 8 bit banyak digunakan di berbagai aplikasi semisal pembaca dan penulis SIMCARD, kunci pintu rahasia, pengendali jarak jauh, penyimpan data, transmisi data melalui gelombang radio dan sebagainya. Beberapa aplikasi membutuhkan keamanan data ketika melakukan proses transfer data atau penyimpanan data.

Implementasi enkripsi/dekripsi pada mikrokontroler 8 bit sangat dibatasi oleh kemampuan sumber daya mikrokontroler tersebut terutama kapasitas memori dan kecepatan eksekusi. Kapasitas memori membatasi panjang kode program yang diimplementasikan sedangkan kecepatan eksekusi membatasi perolehan data terenkripsi perdetiknya. Algoritma enkripsi/dekripsi AES (Rijndael) mempunyai keunggulan karena sebagian besar operasinya berdasarkan data 8 bit sehingga dapat diimplementasikan pada mikrokontroler 8 bit dengan efisien.

Salah satu mikrokontroler AVR adalah seri ATMega 8. Mikrokontroler ini merupakan mikrokontroler 8 bit dengan arsitektur RISC dengan memori program 4 kword (8 kbyte), 1 kbyte RAM. Kecepatan eksekusinya sampai 16 MIPS pada *clock* 16 MHz. Kemasan mikrokontroler ini salah satunya adalah DIP 20 sehingga cukup kecil untuk implemetasi pada aplikasi-aplikasi rumahan (*home application*).

1.2 Rumusan Masalah

Implementasi rutin enkripsi dan dekripsi pada mikrokontroler 8 bit harus memperhatikan sumber daya yang dimiliki agar dapat memenuhi batasan yang dimiliki terutama kapasitas memori dan kecepatan eksekusi. Oleh karena itu rumusan masalahnya adalah bagaimana cara mengimplementasikan algoritma enkripsi dan dekripsi AES (Rijndael) pada mikrokontroler ATMEL AVR ATMega 8 agar memenuhi batasan kapasitas memori dan kecepatan eksekusi.

1.3 Batasan Masalah

Pembahasan pada makalah ini dibatasi pada:

- implementasi rutin enkripsi dan dekripsi AES (Rijndael) untuk masukan dan keluaran 128 bit dengan kunci rahasia 128 bit,
- pengujian rutin enkripsi dan dekripsi AES (Rijndael) menggunakan simulator AVR Studio 4.7 untuk target mikrokontroler AVR ATMega 8,
- data pengujian menggunakan data *testbench* dari Dr Brian Gladman baik masukan maupun keluaran dari masing-masing transformasi.[2]

2 DASAR TEORI

2.1 Spesifikasi Advanced Encryption Standard (AES) Rijndael

Pada awal Agustus 1999, NIST memilih lima algoritma yang akan digunakan sebagai standar, yaitu Mars, RC6, Rijndael, Serpent, dan Twofish. Dari kelima kandidat tesebut Rijndael ditetapkan sebagai standar pada tahun 2001[3].

2.1.1 Masukan, Keluaran, dan Kunci Rahasia

Pada algoritma Rijndael masukan, keluaran, dan kunci rahasia merupakan runtun bit yang terdiri atas 128, 192, atau 256 bit dengan ketentuan panjang masukan dan keluarannya sama. AES menetapkan masukan dan keluaran sebesar 128 bit sedangkan untuk kunci rahasia baik AES dan Rijndael memperbolehkan ketiganya.[2]

2.1.2 Byte

Pada algoritma Rijndael byte terdiri atas 8 bit yang dapat berupa 0 atau 1. Byte merupakan satuan dasar pada semua operasi *cipher*. Untuk operasi cipher, byte dapat dinyatakan dalam runtun bilangan Biner, Heksadesimal, atau dalam bentuk Polinomial yang diintepretasikan sebagai elemen medan terbatas (*finite field elements*).

Contoh:

- {01110011} ; notasi bentuk biner senilai 115
- {73} ; notasi bentuk Heksadesimal senilai 115
- $X^6 + X^5 + X^4 + X + 1$; bentuk polynomial senilai 115

2.1.3 Larik Byte

Masukan, keluaran, dan kunci rahasia merupakan runtun bit yang dinyatakan sebagai larik byte satu dimensi. Byte ke n berisi bit dari 8n sampai dengan 8n+7. Pada masing-masing byte, bit ke 8n+i menjadi bit ke 7-i pada byte yang bersangkutan dengan $0 \le i < 8$. Runtun bit berawal dari kiri ke kanan dimulai dari nol.

Contoh:

Runtun bit:

01001101 10111001 10101001 10100111 11110010

byte 0 byte 1 byte 2 byte 3 byte 4

byte 0 : bit ke 0 s/d bit ke 7

byte 1 : bit ke 8 s/d bit ke 15

byte 2 : bit ke 16 s/d bit ke 23

byte 3 : bit ke 24 s/d bit ke 31

byte 4 : bit ke 32 s/d bit ke 39

kemudian pada masing masing byte, bit ke 8n+i dipetakan ke 7-i

contoh:

byte 2:

bit ke 16 menjadi bit ke 7

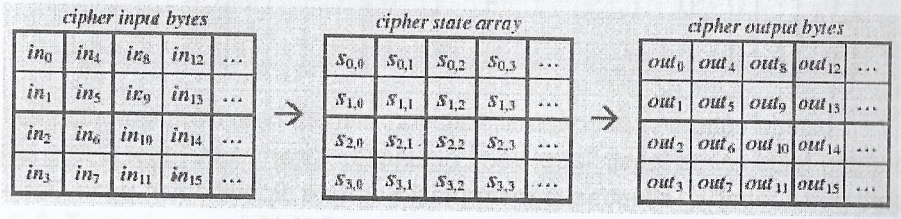
bit ke 17 menjadi bit ke 6

bit ke 18 menjadi bit ke 5, dst

Jadi pada masing-masing byte, paling kiri merupakan bit yang paling berbobot (MSB) dan paling kanan merupakan bit yang paling tidak berbobot (LSB).

2.1.4 *State* AES (Rijndael)

Secara internal algoritma AES (Rijndael) beroperasi dengan menggunakan larik byte dua dimensi yang disebut *state* yang terdiri atas empat baris dan **Nc** kolom, dengan **Nc** merupakan hasil bagi dari panjang masukan (*message block*) dengan 32. Setiap *state* dialamati dengan baris dan kolom, baris dinyatakan dengan r dan kolom dinyatakan dengan c, jadi setiap *state* akan berbentuk seperti s[r,c]. Nilai r bernilai $0 \le r \le 3$ dan c bernilai $0 \le c \le Nc$. Masukan *cipher* yang berbentuk larik satu dimensi dikopikan ke dalam bentuk *state* dan dikeluarkan kembali dalam bentuk larik satu dimensi dengan susunan seperti pada Gambar 2.1.



Gambar 2.1 *Masukan, State dan Keluaran* .[1][2]

2.2 Transformasi Pada Proses Enkripsi AES

Ada empat transformasi yang digunakan pada proses enkripsi, yaitu Transformasi SubBytes, Transformasi ShiftRows, Transformasi MixColumms, dan Transformasi XorRoundKey.

2.2.1 Transformasi SubBytes

Transformasi SubBytes merupakan substitusi byte tidak linier yang dilakukan pada setiap byte *state* untuk menghasilkan nilai baru. Transformasi SubByte terdiri atas dua transformasi yaitu:

1. perkalian inverse pada GF(256), dan
2. *affine transformation* pada GF(2)

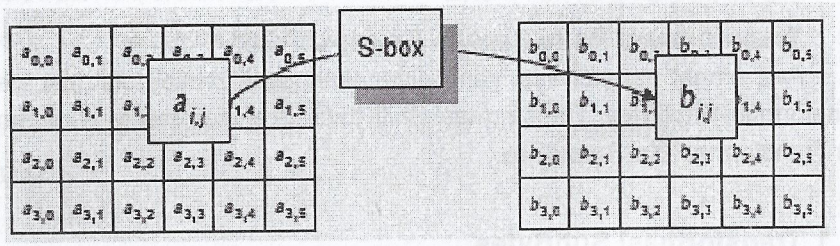
dua transformasi tersebut didefinisikan sebagai berikut:

$$b_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (2.1)^{[2]}$$

dengan c bernilai 0x63. Bentuk matriknya adalah sebagai berikut.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2.2)^{[2]}$$

Untuk mempercepat operasi, transformasi SubByte diimplementasikan dengan teknik baca tabel. Tabel yang digunakan dinamakan Tabel S-Box seperti pada Tabel 2.1. Ilustrasi transformasi SubByte diperlihatkan pada Gambar 2.2.



Gambar 2.2 *Ilustrasi Substitusi SubByte* [1]

Tabel 2.1 *Tabel Substitusi S-box (S-box[xy] dengan xy dalam Heksadesimal)*[2]

hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

x

2.2.2 Transformasi ShiftRows

Trasformasi ShiftRows dilakukan pada tiga baris terakhir, yaitu baris 1, 2, dan 3 (baris dimulai dari 0) menggunakan pergeseran siklis sebagai berikut:

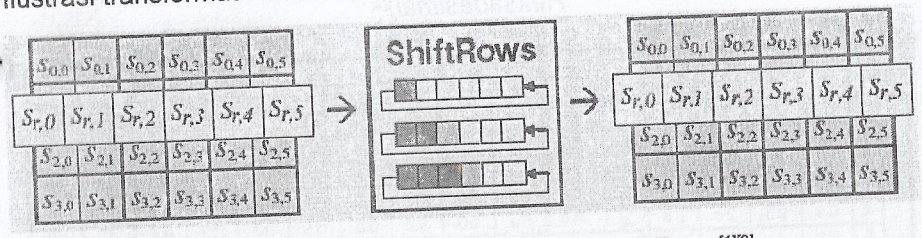
$$S'_{r,c} = S_r = S_{r,\,[c+h(r,Nc)]\bmod Nc} \text{ untuk } 0 \le c < Nc \text{ dan } 0 < r < 4 \quad (2.3)^{[1]}$$

dengan h(r,Nc) tergantung pada jumlah baris (r) dan panjang blok (Nc) seperti pada Tabel 2.2.

Tabel 2.2 *Jumlah Pergeseran h(r,Nc)*[1][2]

h(r, Nc)		*row (r)*		
		1	2	3
Nc	4	1	2	3
	6	1	2	3
	8	1	3	4

Ilustrasi transformasi ShiftRows diperlihatkan pada Gambar 2.3.



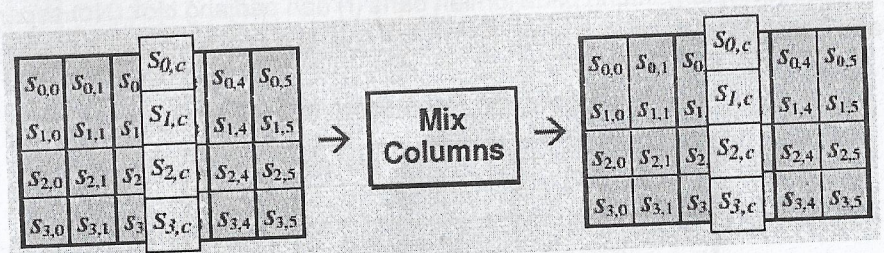
Gambar 2.3 *Ilustrasi Transformasi ShiftRows*[1][2]

2.2.3 Transformasi MixColums

Transformasi MixColums bekerja secara mandiri pada setiap kolom pada *state*. Transformasi MixColums dihitung berdasarkan matriks polinomial sebagai berikut:

$$\begin{bmatrix} s'_{3,c} \\ s'_{2,c} \\ s'_{1,c} \\ s'_{0,c} \end{bmatrix} = \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix} \begin{bmatrix} s_{3,c} \\ s_{2,c} \\ s_{1,c} \\ s_{0,c} \end{bmatrix} \quad \text{for } 0 \le c < Nc \qquad (2.4)^{[1][2]}$$

Ilustrasi transformasi MixColums diperlihatkan pada Gambar 2.4.



Gambar 2.4 *Ilustrasi Transformasi MixColums*[1][2]

FORMAT

2.2.4 Transformasi XorRoundKey

Pada transformasi XorRoundKey Nc kolom pada hasil ekspansi kunci rahasia dijumlahkan (di-XOR-kan) dengan kolom pada *state* sehingga

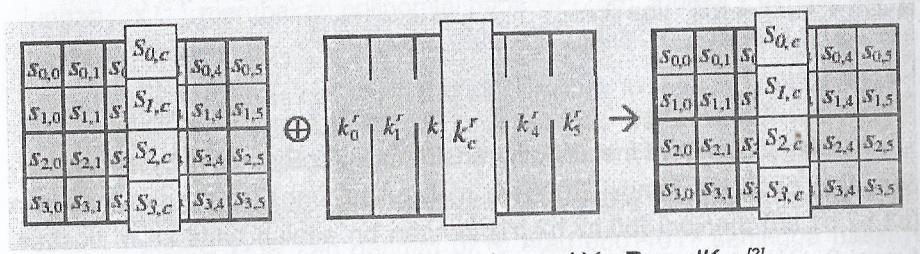
$[b'_{3c'}\, b'_{2c'}\, b'_{1c'}\, b'_{0c'}] = [b_{3c'}\, b_{2c'}\, b_{1c'}\, b_{0c'}]$ XOR $[k_{round,c}]$ untuk $0 \le c < Nc$ (2.5)[2]

$[k_{round},c]$ adalah kunci rahasia yang telah diekspansi, *round* bernilai $0 \le$ round $\le$ Nn. Nol merupakan inisial *round* yang digunakan sebelum iterasi *round*. Nn adalah jumlah *round* yang nilainya ditentukan dengan panjang blok (Nc) dan kunci rahasia (Nk) seperti Tabel 2.3.

Tabel 2.3 *Jumlah Round (Nn) Sesuai Jumlah Nc dan Nk*[1][2]

Nn		Nc		
		4	6	8
Nk	4	10	12	14
	6	12	12	14
	8	14	14	14

Ilustrasi transformasi XorRoundKey diperlihatkan pada Gambar 2.5.

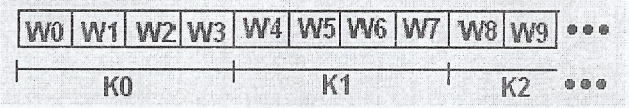


Gambar 2.5 *Ilustrasi Transformasi XorRoundKey*[2]

2.3 Ekspansi Kunci Rahasia

Kunci rahasia diekspansi secara terpisah dan dilakukan pada saat awal, sebelum dilakukan proses enkripsi dan dekripsi. Kunci rahasia yang diekspansi digunakan pada proses enkripsi secara maju dan pada proses

dekripsi secara mundur. Susunan hasil ekspansi kunci rahasia diperlihatkan seperti Gambar 2.6.



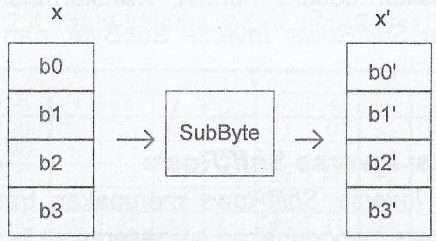
Gambar 2.6 *Susunan Hasil Ekspansi Kunci Rahasia*[1][2]

Wn adalah kolom kunci rahasia hasil ekspansi yang terdiri atas empat buah baris. Kn adalah kunci rahasia pada *round* ke n. Kn berisi 4 *word* dari W[Nb*i] sampai W[Nb*(i+1)].

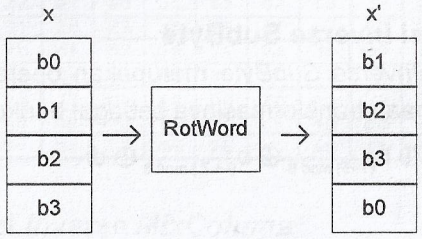
Pseudocode ekspansi kunci rahasia adalah sebagai berikut[1][2]:

```
KeyExpansion(byte Key[ 4*Nk] , word W[ Nb* (Nr+1)] )
{
for(i = 0; i < Nk; i++)
       W[ i] = (Key[ 4*i] ,Key[ 4*i+1] ,Key[ 4*i+2] ,Key[ 4*i+3] );
for(i = Nk; i < Nb * (Nr + 1); i++)
{
       temp = W[ i - 1] ;
if (i % Nk == 0)
       temp = SubWord(RotByte(temp)) XOR Rcon[ i / Nk] ;
W[ i] = W[ i - Nk] XOR temp;
}
}
```

Fungsi SubWord melakukan transformasi SubByte pada byte yang bersesuaian pada kn. Fungsi RotWord melakukan fungsi rotasi dari masukan [b3,b2,b1,b0] menjadi [b0,b3,b2,b1] dengan bn adalah baris ke n. Ilustrasi fungsi SubWord dan RotWord diperlihatkan pada Gambar 2.7 dan 2.8.



Gambar 2.7 *Ilustrasi Fungsi SubWord*



***Gambar 2.8** Ilustrasi Fungsi RotWord*

Sedangkan larik Rcon[y] dengan y = i/Nk berisi nilai [0, 0, 0, x^{y-1}] dengan x^{y-1} merupakan polinomial pada GF(256). Indeks y dimulai dari 1. Contoh:

```
y = 1, berarti x ⁰ dalam bentuk biner =  00000001
y = 2, berarti x ¹ dalam bentuk biner =  00000010
y = 9, berarti x ⁸ dalam bentuk biner = 100000000
```

(karena keluar dari GF(256) maka perlu dikoreksi dengan cara meng- XOR kan dengan 0x1b atau 00011011 sehingga hasilnya untuk y=9 adalah 00011011).

2.4 Transformasi pada Dekripsi

Transformasi antara enkripsi dan dekripsi sebagian besar berbeda, hanya transformasi *XorRoundKey* saja yang sama. Hal ini merupakan salah satu kelemahan ketika implementasi karena kode program enkripsi dan dekripsi

harus diimplementasikan secara mandiri. Transformasi dekripsi terdiri atas transformasi *Inverse ShiftRows, Inverse SubByte*, dan *Inverse MixColums* dan *XorRoundKey.*

2.4.1 Transformasi *Inverse ShiftRows*

Transformasi *Inverse ShiftRows* merupakan transformasi balik dari transformasi *ShiftRows* menggunakan pergeseran siklis sebagai berikut:

$$S'_{r,[c+h(r,Nc)] \bmod Nc} = S_{r,c} \text{ untuk } 0 \le c < Nc \text{ dan } 0 < r < 4 \qquad (2.6)^{[1][2]}$$

2.4.2 Transformasi Inverse SubByte

Transformasi *Inverse SubByte* merupakan operasi balikan dari Sub-Byte dengan persamaan transformasinya sebagai berikut.

$$b_i' = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus di \qquad (2.7)^{[2]}$$

dengan d=0x05

Transformasi Inverse SubByte diimplementasikan menggunakan tabel inversi S-Box (Tabel 2.4).

Tabel 2.4 *Tabel Inverse S-Box (InvS-Box[xy])*[1][2]

hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

(x)

2.4.3 Transformasi *Inverse MixColums*

Transformasi *inverse MixColums* menggunakan matriks sebagai berikut:

$$\begin{bmatrix} s'_{3c} \\ s'_{2c} \\ s'_{1c} \\ s'_{0c} \end{bmatrix} = \begin{bmatrix} 0e & 09 & 0d & 0b \\ 0b & 0e & 09 & 0d \\ 0d & 0b & 0e & 09 \\ 09 & 0d & 0b & 0e \end{bmatrix} \begin{bmatrix} s_{3c} \\ s_{1c} \\ s_{1c} \\ s_{0c} \end{bmatrix} \text{ for } 0 \le c < Nc \qquad (2.8)^{[1][2]}$$

2.5 Algoritma Enkripsi

Algoritma enkripsi AES dimulai dengan mengkopi masukan enkripsi (*block message*) ke *internal state*, kemudian dilakukan proses XOR dengan initial kunci rahasia (*initial round*), selanjutnya dilakukan proses *round* dengan iterasi berdasarkan panjang *block message* dan kunci rahasia, dan diakhiri dengan *final round*. Hasil keluaran dapat dikopikan ke *state* keluaran. *Pseudo-code* algoritma AES seperti berikut ini:

```
Enkripsi(State,EkspanKey)
{
XorRoundKey(state, K0)
for (round = 1; round<10;round++)
       {
       SubBytes(state)
       ShiftRows(state)
       MixColumns(state)
       XorRoundKey(state, K(round))
       }
SubBytes(state)
ShiftRows(state)
XorRoundKey(state, K10)
}
```

2.6 Algoritma Dekripsi

Proses dekripsi melakukan proses balik, masukan *chiper* diproses dengan kunci rahasia secara mundur. *Pseudocode* algoritma dekripsi adalah sebagai berikut:

```
Dekripsi(State,EkspanKey)
{
XorRoundKey(state, K10)
for (round =9; round>0;round--)
  {
   InvShiftRows(state)
   InvSubBytes(state)
   XorRoundKey(state, K(round))
   InvMixColumns(state)
  }
InvShiftRows(state)
InvSubBytes(state)
XorRoundKey(state)
}
```

2.7 Mikrokontroler ATMEL AVR ATMega 8

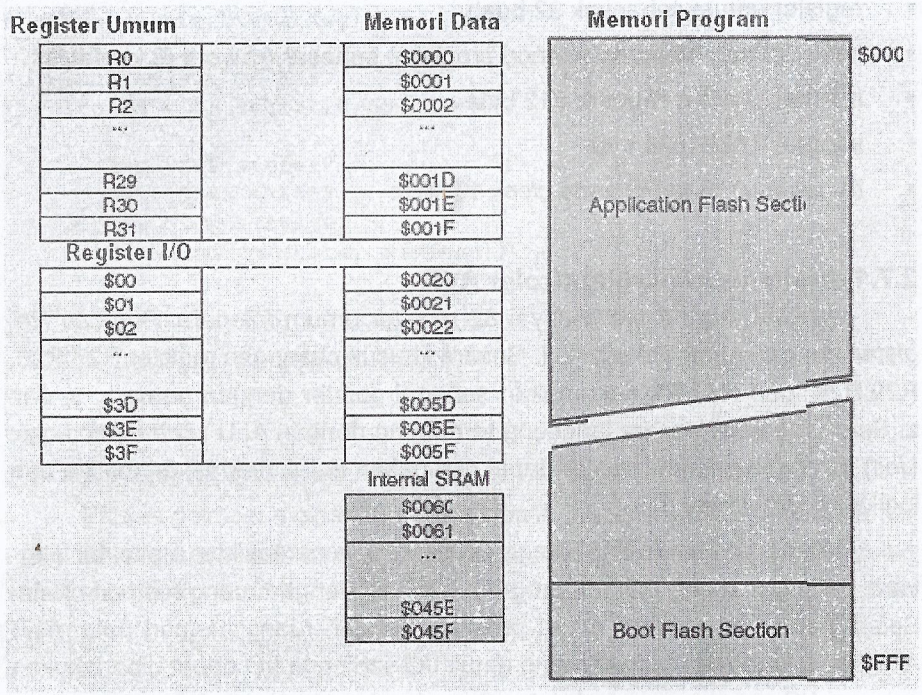
Mikrokontroler ATMEL AVR ATMega 8 merupakan mikrokontroler berbasis RISC yang diproduksi oleh ATMEL. Spesifikasi umum dari mikrokontroler ini adalah:

- register umum sebanyak 32 buah
- internal *flash memory* (memori program) sebesar 4 kword (8 kbyte)
- internal eeprom sebesar 512 byte
- internal RAM 1024 byte
- *throughput* 16 MIPS pada *clock* 16 MHz

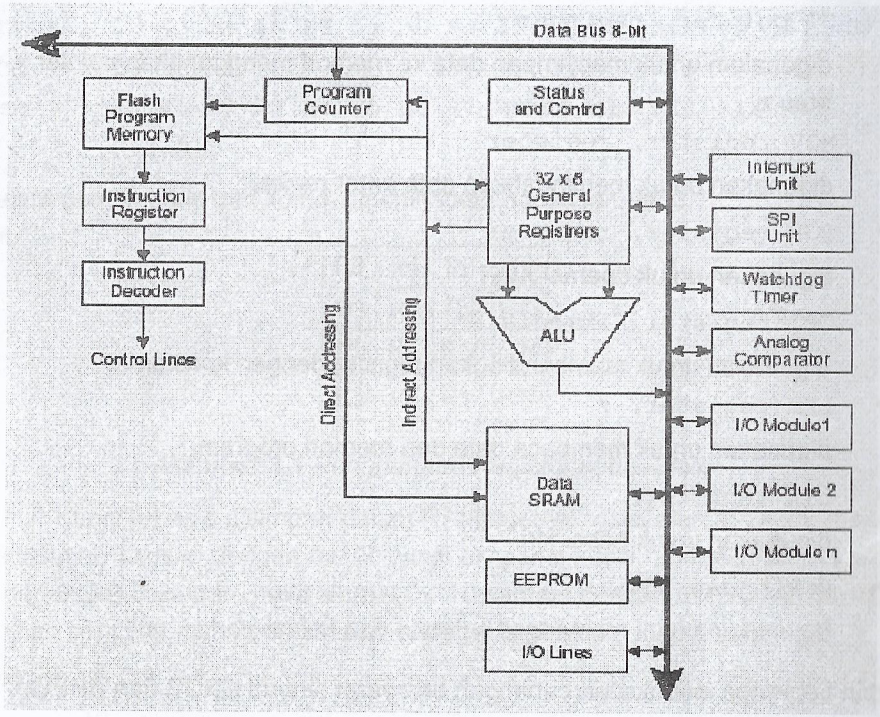
2.7.1 Arsitektur Mikrokontroler AVR

AVR ATMega 8 mempunyai 32 register umum (*General Purpose Register* dengan nama R0 s/d R31. Secara khusus pasangan register R27:R26, R29:R28, dan R31:R30 digunakan sebagai pointer dengan nama x, y, dan z. Semua register secara langsung terhubung dengan ALU (*Arithmatic Logic Unit*) sehingga semua register dapat digunakan untuk operasi aritmatika dan logika secara langsung.

Mikrokontroler AVR ATMega 8 mengimplementasikan arsitektur Harvard, sehingga ruang memori program terpisah dengan ruang memori ridata. Selain pemisahan antara ruang program dengan ruang memori data, AVR ATMega 8 juga mempunyai ruang untuk I/O sehingga I/O dapat diperlakukan sebagai I/O maupun memori data. Organisasi memori dan I/O untuk mikrokontroler AVR ATMega 8 diperlihatkan pada Gambar 2.9. Sedangkan arsitektur intinya diperlihatkan pada Gambar 2.10.



Gambar 2.9 *Organisasi Memori ATMega 8*



Gambar 2.10 *Arsitektur Inti Mikrokontroler AVR AT Mega 8*

2.7.2 Set Instruksi

Beberapa instruksi yang digunakan antara lain:

- `LDI register, immediate`

  digunakan untuk memberi nilai register. Register yang diperbolehkan adalah R16 s/d R31.

- `LD register, pointer`

  digunakan untuk membaca data dari memori secara tak langsung menggunakan pointer x, y, atau z.

- `ST pointer, register`
  digunakan untuk menyimpan data ke memori menggunakan pointer x, y, atau z.
- `MOV register, register`
  digunakan untuk memindahkan data antar register.
- `EOR register, register`
  digunakan untuk operasi XOR .
- `CPI register, immediate`
  digunakan untuk membandingkan register dengan konstanta.
- `LPM register, z`
  digunakan untuk membaca data dari memori program.
- `LSL register`
  digunakan untuk geser kiri.
- `RCALL dan RET`
  digunakan untuk memanggil subrutin dan keluar dari subrutin.

dan beberapa instruksi percabangan bersyarat seperti BRCC dan BREQ.

3 IMPLEMENTASI

3.1 Penggunaan Bahasa Pemrograman dan *Tools*

Rutin enkripsi dan dekripsi diimplementasikan menggunakan bahasa asembli AVR. Sedangkan editor, asembler, dan simulator menggunakan AVR Studio 4.7 dari ATMEL.

3.2 Penggunaan Register dan Memori

Masukan *plaintext*, *state*, dan keluaran *chiper* menggunakan register yang sama yaitu R0 s/d R15. Sehingga sebelum memanggil subrutin enkripsi dan dekripsi, blok masukan *plaintext* dikopikan ke dalam register R0 s/d R15 dengan urutan sebagai berikut:


```
Input : 32 43 f6 a8 88 5a 30 8d 31 31 98  a2  e0  37  07
        34
reg   : R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13
        R14 R15
```

dalam bentuk baris dan kolom diperlihatkan pada Gambar 3.1.

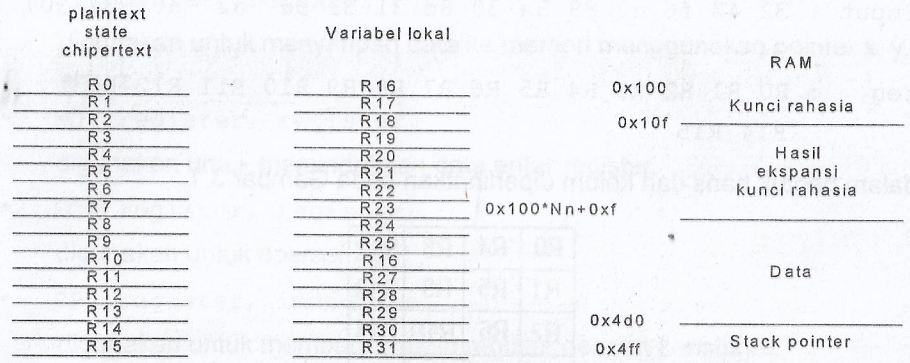
R0	R4	R8	R12
R1	R5	R9	R13
R2	R6	R10	R14
R3	R7	R11	R15

Gambar 3.1 *Penggunaan Register untuk State*

Kunci rahasia disimpan dalam RAM internal mulai alamat 0x100 (heksadesimal) sampai dengan 0x10f. Hasil ekspansi kunci rahasia (round 1 s/d round 0x0a) disimpan mulai alamat 0x110 sampai dengan 0x110 * 0x0a + f. Urutan penyimpanan kunci rahasia sebagai berikut:

```
Key = 2b 7e 15  16 28  ae d2 a6 ab f7 15 88  09  cf  4f
      3c
alm   0x100           s/d                          0x10f
```

Register R16 s/d R31 digunakan untuk variabel lokal pada subrutin-subrutin di bawahnya. Seluruh operasi mengeksploitasi penggunaan register agar didapatkan kecepatan operasi yang tinggi. Ilustrasi penggunaan register dan memori diperlihatkan pada Gambar 3.2.



Gambar 3.2 *Ilustrasi Penggunaan Register dan RAM Internal*

3.3 Rutin Enkripsi dan Dekripsi

Rutin enkripsi terdiri atas subrutin-subrutin *initial round, iteration round,* dan *final round* sebagai berikut:

```
rijndael_enkrip:
        rcall initial_round
        rcall sembilan_round
        rcall final_round
        ret
```

Pada subrutin initial_round melakukan operasi XOR antara *plaintext* dengan kunci rahasia (K0) (*exorkey*). Subrutin sembilan_round melakukan operasi *subbyte, shift row, mixcolm,* dan *xorkey* dengan K1 sampai dengan K9. Subrutin final_round melakukan operasi *subbyte*, *shift row*, dan *xorkey* dengan K10.

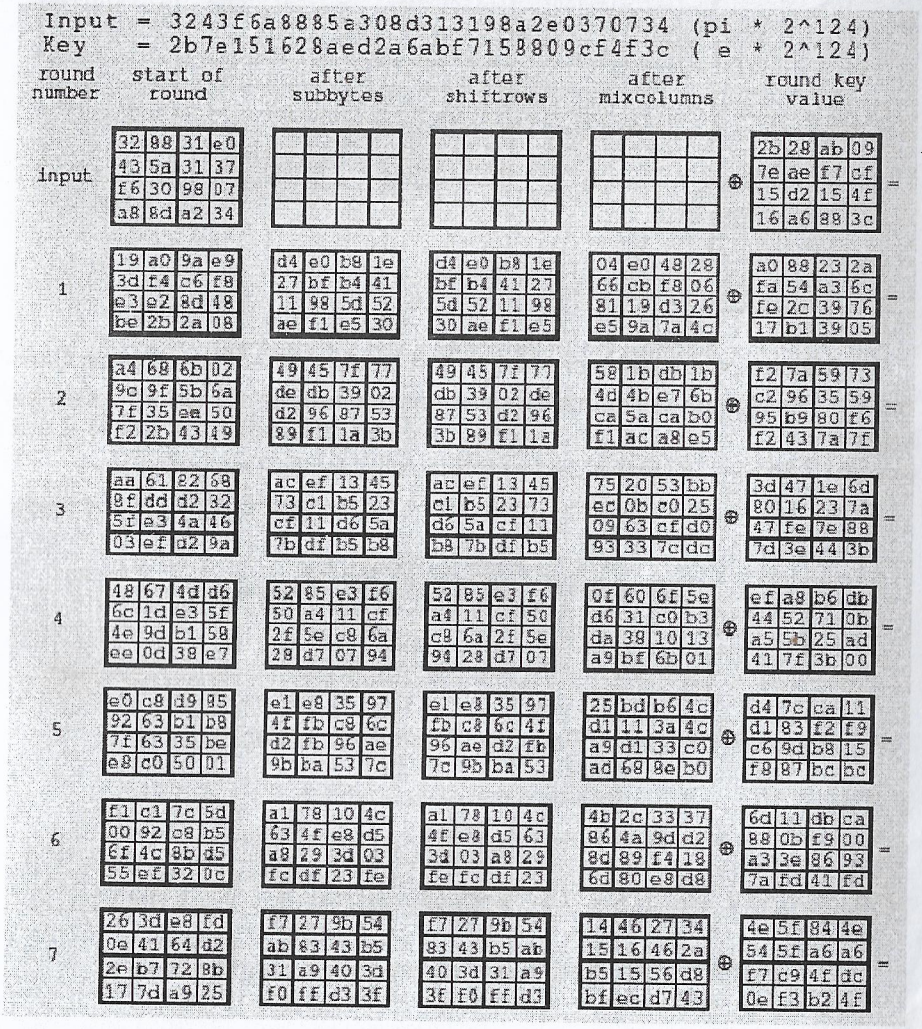
Sedangkan rutin dekripsi akan melakukan fungsi sebaliknya:

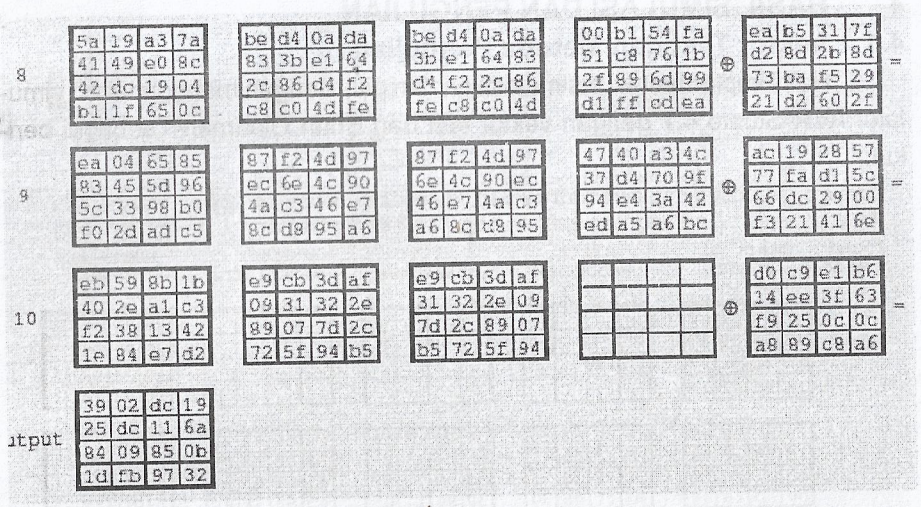
```
rijndael_dekrip:
        rcall inv_initial_round
        rcall inv_sem_round
        rcall inv_final_round
        ret
```


4 HASIL SIMULASI DAN PENGUJIAN

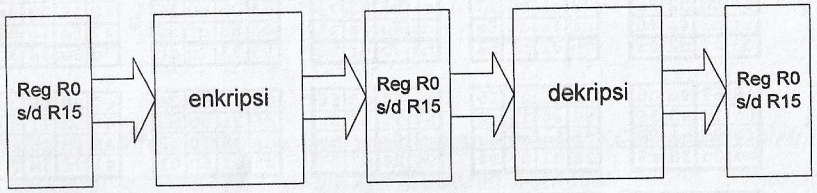
4.1 Vektor Test dan Metoda Pengujian

Hasil implementasi disimulasikan dan diuji dengan menggunakan simulator AVR Studio 4.7 dengan vektor test dari Brian Gladman[2] sebagai berikut:





Metode pengujian rutin enkripsi dan dekripsi diperlihatkan pada Gambar 4.1. Hasil pada setiap transformasi disimpan pada R0 s/d R15 sebagai state.

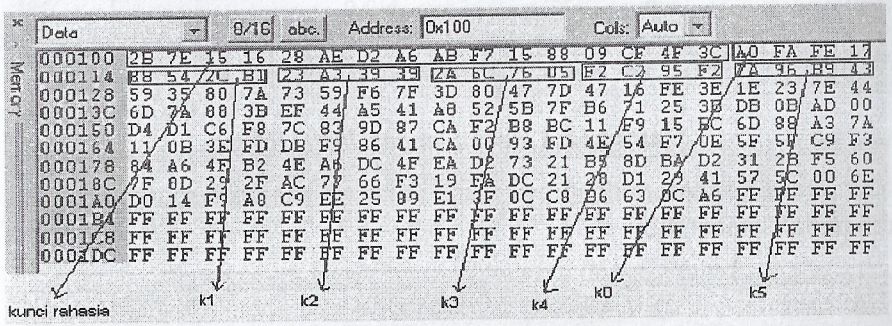


Gambar 4.1 *Metode Pengujian Rutin Enkripsi dan Dekripsi*

4.2 Hasil Simulasi dan Pengujian

4.2.1 Hasil Ekspansi Kunci Rahasia

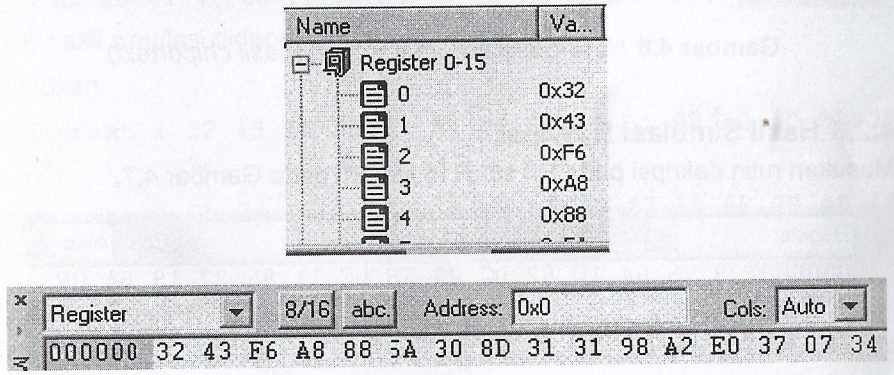
Hasil ekspansi kunci rahasia diperlihatkan pada Gambar 4.2.



Gambar 4.2 *Hasil Ekspansi Kunci Rahasia*

4.2.2 Hasil Simulasi Enkripsi

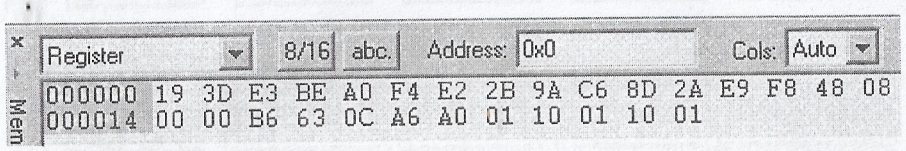
Hasil simulasi Plaintext dimasukkan ke R0 s/d R15 (128 bit / 16 byte) diperlihatkan pada Gambar 4.3.



Gambar 4.3 *Plaintext Dimasukkan ke R0 s/d R15*

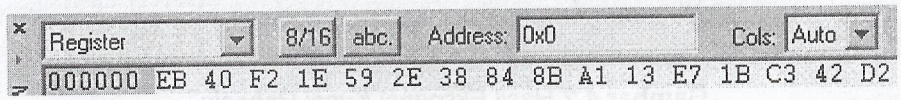
Register R0 paling kiri dengan alamat 0x000000 dan R15 paling kanan pada alamat 0x00000f.

Hasil simulasi pada initial round diperlihatkan pada Gambar 4.4.



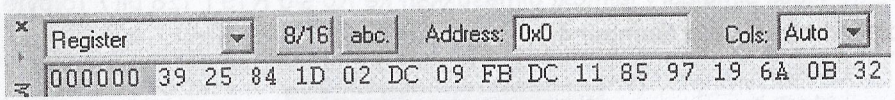
Gambar 4.4 *Hasil Simulasi Initial Round*

Hasil simulasi pada iterasi round (subrutin sembilan round) diperlihatkan pada Gambar 4.5.



Gambar 4.5 *Hasil Simulasi Iterasi Round*

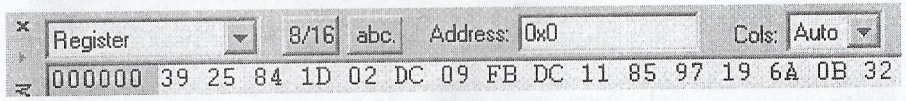
Hasil simulasi pada *final round* (*chipertext*) diperlihatkan pada Gambar 4.6.



Gambar 4.6 *Hasil Simulasi Final Round (hasil chipertext)*

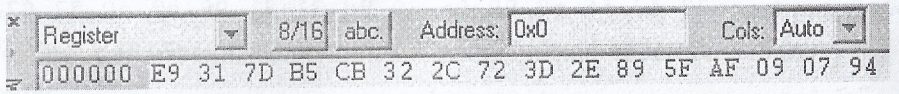
4.2.3 Hasil Simulasi Dekripsi

Masukan rutin dekripsi pada R0 s/d R15 seperti pada Gambar 4.7.



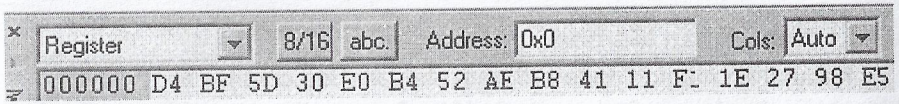
Gambar 4.7 *Masukan Rutin Dekripsi*

Hasil simulasi *inverse* initial *round* diperlihatkan pada Gambar 4.8.



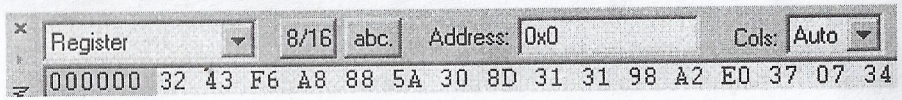
Gambar 4.8 *Hasil Simulasi Inverse Initial Round*

Hasil simulasi *inverse* iterasi *round* (inv sembilan *round*) diperlihatkan pada Gambar 4.9.



Gambar 4.9 *Hasil Simulasi Inverse Iterasi Round*

Hasil simulasi *inverse final round* (hasil *plaintext*) diperlihatkan pada Gambar 4.10.



Gambar 4.10 *Hasil Simulasi Inverse Final Round*

Dari hasil simulasi didapatkan:

Masukan:

```
plaintext : 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07
            34
key       : 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f
            3c
```

Keluaran Enkripsi (Masukan Dekripsi):

```
chipertext : 39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b
             32
```

Keluaran Dekripsi:

```
plaintext  : 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07
             34
```

4.2.4 Hasil Simulasi Kinerja

Hasil simulasi kinerja dari rutin enkripsi dan dekripsi yang diimplementasikan pada mikrokontroler ATMEL AVR ATMega 8 adalah sebagai berikut:

4.2.4.1 Kebutuhan Memori

- Kebutuhan memori pada program ekspansi kunci rahasia:

```
Segment   Begin     End       Code   Data   Used   Size   Use%
------------------------------------------------------------
[ .cseg] 0x000000 0x00023a    314    256    570   8192   7.0%
[ .dseg] 0x000060 0x000060      0      0      0   1024   0.0%
[ .eseg] 0x000000 0x000000      0      0      0    512   0.0%
```

- Kebutuhan memori pada program enkripsi:

```
Segment   Begin     End       Code   Data   Used   Size   Use%
------------------------------------------------------------
[ .cseg] 0x000000 0x0002fa    506    256    762   8192   9.3%
[ .dseg] 0x000060 0x000060      0      0      0   1024   0.0%
[ .eseg] 0x000000 0x000000      0      0      0    512   0.0%
```

- Kebutuhan memori pada program dekripsi:

```
Segment   Begin     End       Code   Data   Used   Size   Use%
------------------------------------------------------------
[ .cseg] 0x000000 0x0004c2    962    256   1218   8192  14.9%
[ .dseg] 0x000060 0x000060      0      0      0   1024   0.0%
[ .eseg] 0x000000 0x000000      0      0      0    512   0.0%
```

- Kebutuhan memori pada program enkripsi dan dekripsi:

```
Segment   Begin     End       Code   Data   Used   Size   Use%
------------------------------------------------------------
[ .cseg] 0x000000 0x0006f0    1264    512   1776   8192  21.7%
[ .dseg] 0x000060 0x000060      0      0      0   1024   0.0%
[ .eseg] 0x000000 0x000000      0      0      0    512   0.0%
```


Implementasi program enkripsi, dekripsi, dan ekspansi kunci rahasia lebih efisien diimplementasikan secara bersama karena program enkripsi, dekripsi, dan ekspansi kunci rahasia menggunakan beberapa subrutin yang sama.

4.2.4.2 Kecepatan Eksekusi

- Kecepatan (*cycle*):

```
ekspansi kunci rahasia (key)    = 2715 cycle
```

enkripsi:

```
- persiapan          =   39 cycle
- initial_round      =   64 cycle
- 9 iterasi_round    = 6625 cycle
- final_round        =  510 cycle
- total              = 7238 cycle
```

dekripsi:

```
- persiapan          =   39 cycle
- inv_initial_round  =   64 cycle
- inv_iterasi_round  =14848 cycle
- inv_final_round    =  510 cycle
- total              =15461 cycle
```

Satu *cycle* pada mikrokontroler AVR ATMega 8 sama dengan perioda kristal (osilator) yang terpasang pada mikrokontroler tersebut. Contoh, kristal yang digunakan adalah 4 MHz maka perioda (T) = 25 us sehingga 1 *cycle* sama dengan 25 us. Kecepatan eksekusi ekspansi kunci rahasia, subrutin enkripsi dan dekripsi dalam detik adalah:

```
- ekspansi kunci rahasia =  2715 x 25 us =  67875 us
                                         =  67,875 m
- enkripsi               =  7238 x 25 us = 180950 us
                                         = 180,95 ms
```

- dekripsi = 15461 x 25us = 386525 us
  = 386,525 ms

5 KESIMPULAN

Dari hasil pembahasan pada bab sebelumnya dapat diambil kesimpulan sebagai berikut:

- enkripsi/dekripsi AES (Rijndael) dapat diimplementasikan pada mikrokontroler ATMEL AVR ATMega 8 karena hanya membutuhkan 21,7% dari kapasitas memori program yang dimiliki.
- algoritma dekripsi membutuhkan ruang memori yang lebih banyak dan waktu eksekusi yang lebih lama dari pada algoritma enkripsi.
- subrutin iterasi_round dan inv_iterasi_round memerlukan waktu eksekusi yang paling besar pada masing-masing algoritma.

6 DAFTAR PUSTAKA

1. J. Daemen, Vincent Rijmen, September 3, *AES Proposal: Rijndael, AES Algorithm Submission*, 1999, NIST. http://csrc.nist.gov/encryption/aes
2. Dr Brian Gladman, May 2002, *A Specification for Rijndael, the AES Algorithm*.
3. Sungha Kim, Ingrid Verbauwhede, September 2002, *AES Implementation on 8-bit Microcontroller*, Department of Electrical Engineering University of California, Los Angeles, USA.
4. Datasheet, 2005, *8 bit Mikrokontroler With 16 Kbyte In-System Programming Flash ATMega8*, ATMEL Co, www.atmel.com.

INTERNAL INTEGRATED MARKETING STRATEGI UNTUK MERAIH KEUNGGULAN BERSAING JANGKA PANJANG

Oleh: Budi Sugiharjo

ABSTRAK

Persaingan bisnis di era yang penuh gejolak bahkan cenderung turbulen ini, menjadikan banyak macam strategi pemasaran klasik yang bersifat parsial tidak lagi mampu mendatangkan hasil maksimal. Di tengah-tengah situasi di mana masyarakat begitu mudah mendapatkan informasi, maka konsumen digambarkan sebagai sosok manusia yang penuh dengan informasi dan pengetahuan. Konsumen tidak lagi bisa dipengaruhi dengan berbagai janji iklan yang tidak memberikan bukti. Secara psikologis kini masyarakat juga tidak senang jika mengetahui bahwa dirinya sedang disegmenkan untuk selanjutnya dibidik menjadi target pasar usaha penjualan suatu produk perusahaan. Model pemasaran dengan pendekatan parsial, dan memposisikan konsumen atau calon konsumen sebagai pihak luar dan pihak lain atau liyan, dipastikan tidak akan mampu lagi digunakan untuk mempertahankan atau mendongkrak volume penjualan jangka panjang. Didasari oleh pemikiran yang demikian, maka penulis menawarkan suatu model pemasaran baru yang disebut dengan "*Internal Integrated Marketing*" sebagai suatu alternatif model pemasaran yang jauh lebih efektif dan efisien untuk usaha mempertahankan atau meningkatkan volume penjualan jangka panjang suatu perusahaan atau institusi non bisnis.

Kata Kunci: Internal, Integrated, Marketing, Pemasaran horizontal, Pemasaran vertikal, stakeholders, dan komunitas.